International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 1; Issue: 8; December–2013

FP-Growth algorithm in Data Compression frequent patterns

Mr. Nagesh V
Lecturer, Dept. of CSE
Atria Institute of Technology,AIKBS
Hebbal , Bangalore ,Karnataka
Email : nagesh.v@gmail.com

Abstract-The transmission of supermarket data from one machine to another demands high bandwidth because, usually it is very vast. In order to reduce the bandwidth requirement, usually we compress the data before transmission and decompress the same in the target machine. In general, the supermarket data has some relationship among the items in a bucket or transaction.This relationship among the items in the transactions can be identified by using an association rule mining algorithms like FP growth. Once we know the frequent patterns of the data items, this can be used to construct the dictionary in order to improve the performance of the data compression. The new data compression algorithm will make use of the frequent patterns, which has been identified by FP-growth algorithm, to construct the static dictionary. This static dictionary will be used by both compression and decompression techniques.

Key words: FP-tree, FP-growth algorithm, Dictionary.

1. Introduction

In Market Basket Analysis the most importent question is " Which group of items are customers likely to buy together?" To answer this question we need to check at the transaction databse of the store and find which set of items are bought most frequently. This gives rise to a widely studied problem called "Frequent Pattern Mining".

A pattern is a set of products or items that is interesting in some way. If a set of items occurs very frequently among the transactions, it is interesting indeed. Thus a pattern that occurs more than a pre specified number of times it is called a frequent pattern. There are several algorithms for mining frequent patterns in the transaction database. Apriori is one of the earliest and mostly studied algorithm. Apriori is based on the anti-monotone Apriori Principle which says that a "if any length K pattern is not frequent in the database, its length k pattern is not frequent in the database, its length (k+1) super-pattern can never be frequent". Based on this principle , a number of algorithms(i.e. MaxMiner, AprioriTID, etc.) are given. There is another class of algorithms uses an extended prefix tree of the transactions called FP-tree. In this report, three different mining approaches of that class are described. FP-growth is the earliest of them and it generates the complete set of frequent patterns. The other two approaches are mining Top K Frequent Closed Pattern and mining Compressed Frequent Pattern. These algorithms are faster than most of the Apriori Based algorithms.

The remaining of the report consists of four sections. In section 2 frequent pattern mining problem is formally defined. In section 3 previous algorithms on frequent pattern mining is discussed. In section 4 the algorithm for constructing the FP- tree is explained. Section 5 describes the algorithm FP-growth. In section 6 and 7, a couple of extensions of the frequent pattern mining problems are drawn and for each problem an algorithmic framework is given. Section 8 and 9 shows some performance studies and related works. And in Section 10 we conclude.

2. Problem Definition

Let I = {a_1, a_2, ..., a_m} is a set of items. A transaction T_I is a set of items representing the items of sale in real world. A transition database DB= T_1, T_2, ..., T_n is a set of Transactions representing sales data over a significant period of time(e.g. a day, a week, etc). A pattern P is a subset of a least one transaction of DB. Support of a pattern P is the number of transactions T_i such that $P \subset T_i$. We will represent the support of a pattern P as P.support. a pattern P is frequent pattern if P.support $>\xi$ where ξ is the minimum support threshold.

For a transaction database DB and minimum support threshold ξ, finding the complete set of frequent patterns is called the frequent pattern mining problem.

3. Previous Algorithms

Apriori is the mostly studied frequent pattern mining algorithm, it starts with the set of frequent items (length=1) that have minimum support. In each iteration it generates a set of candidate(potential) patterns of the next length. Then it checks transaction in the database to count the support for each of the candidate set. It terminates when there is no candidate patterns of next length is achievable. Various strategies for generating the candidate set and updating the count values are developed. All of these strategies have the same disadvantages which are

- Generating candidate set is redundant because many candidates are not actual patterns.
- Generating candidate set is costly because it may end up with exponential number of candidate sets. For example, to generate a pattern of length k it may consist of 2^k candidate sets.
- Scanning the database in each iteration for updating the count values is costly.

To get rid of these problems, we have to give up generating patterns and counting their supports. Here the FP-tree comes into play. FP-tree is a compact data structure that summarizes the whole database and sufficient for growing larger patterns from smaller ones. In the next section, construction and properties of FP-tree are described.

4. FP-tree

Structure:

An FP-tree has two main components. The item prefix subtrees with a root and frequent-item header table.

Each node in an item prefix subtree consists of three fields, item-name, count and node link. Count registers the number of transactions that support the itemset built by accumulating the items from root to that node. Node-link is the link to the next node having the same item-name or null if there is none.

Frequent-item header table has two columns, item-name and head of node-link which points to the start of the list in the FP-tree for its item-name.

Construction:

Before constructing the FP-tree we need to scan the whole database once to find the set F of frequent items(items with support>ξ) and their supports. We then sort F in the descending order of the supports. This sorted list of items is called L.

We then create a root node with no item-name. We then take each transaction at a time to update the tree. For each transaction we sort the items in the order of L and insert the items into the tree in that order. At the time of insertion, we always try to merger the prefix path of the current transaction with an existing path in the FP-tree and create a branch if it does not match.

Figure1 shows a transactions database and Figure 2 shows its FP-tree for ξ=3.

TID	Items Bought	Frequent Items
100	*f,a,c,d,g,i,m,p*	*f,c,a,m,p*
200	*a,b,c,f,l,m,o*	*f,c,a,b,m*
300	*b,f,h,j,o*	*f,b*
400	*b,c,k,s,p*	*c,b,p*
500	*a,f,c,e,l,p,m,n*	*f,c,a,m,p*

Figure1: Transaction database

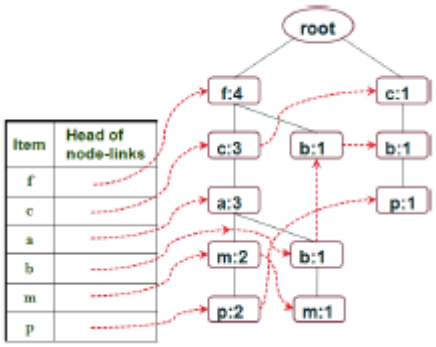

Figure2: FP-tree

Properties:

- FP-tree contains the information of a transaction database completely and compactly for a particular threshold ξ.
- Height of the Fp-tree is bounded by the maximal number of frequent items in any transaction in the database.

Next section describes how we can use the FP-tree to generate the Frequent patterns.

5. FP-growth

FP-growth is an algorithm for finding the complet set of frequent patterns using the tree. The FP-tree tells us that all the items in the tree are frequent. To mine larger patterns, FP-growth builds Conditional Pattern base and then Conditional FP-tree for each item a_i in the item header table. Each pattern in a conditional pattern base for a_i is created by accumulating all the items in the path from root to a node containing the a_i with a support equal to the a_i. After accumulating all the conditional patterns from all the a_i nodes, FP-growth builds the conditional FP-tree as previously described. Conditional FP-tree tells us that if we concatenate any item on it with ai that will be a frequent pattern. Thus FP-growth builds larger patterns in a recursive fashion by creating smaller conditional FP-tree at each step. A sample path of the recursion tree is shown in Figure3.

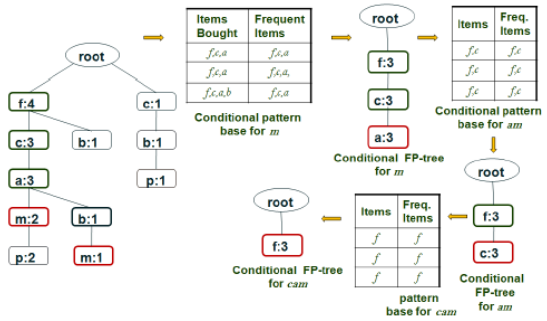

Figure 3: Conditional FP-tree for m

Another observations is that an FP-tree consisting of a path generates all the combinations of its nodes as frequent patterns. Figure-4 shows a single path tree and the patterns generated by this path. Thus by generating all the combinations of the nodes in a path we can save some recursive calls to FP-growth. Including this saving scheme the pseudo code for FP-growth is given in Table1.

To use FP-growth successfully the user must know the minimum support threshold(ξ) for his/her purpose. For a large support threshold FP-growth may generate few patterns of no interest. While a smaller threshold may end up with a large number of frequent patterns. To make it more usable, variations of frequent pattern mining is developed. In the next two sections two of such variations is described.

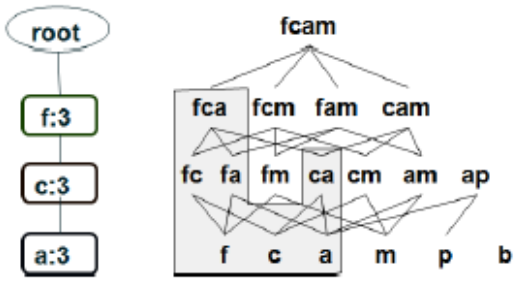

Figure 4. Patterns generated by Single path

```
Procedure FP-growth(Tree,α)
    if Tree contains a single path P
        for each combination (β) of the nodes in P
            Generate pattern β∪α with
              support = minimum support of a node in β
    else
        for each a_i in the header of Tree do
            Generate pattern β = α∪a_i with support = a_i.support.
            Construct βs conditional pattern base
              and conditional FP-tree Tree_β
            if Tree_β ≠ φ
                Call FP-growth(Tree_β, β)
```

Table 1:Pseudo code for FP-growth

6. Top- K frequent closed pattern mining

Top-K frequent closed pattern mining is a mining task where only the top-K patterns in orders of the supports are output. So there is no support threshold here. Besides the ouput of FP-growth may contain a pattern that has some of its subpatterns in the output with same support. These subpatterns are completely included in the super pattern. That's why this approach of mining frequent pattern mines only closed pattern. A closed pattern is a pattern whose support is larger than any of its super pattern. For example in Figure: 5 a complete set of frequent patterns and their supports are shown. The red patterns are closed patterns. If we awant to find Top-2 frequent patterns with length >2 they will be fc and fcam.

To mine Top-K closed pattern, the FP-tree is used. We first build an FP-tree for ξ=0. While building the tree, any transaction with length less than the min-length is pruned out. After the tree is built completely, relatively infrequent patterns can be pruned by raising the minimum support from 0. Two methods are used for this purpose. The closed-node-count and the descendent-sum. After all the pruning is done we can now generate the Top-K closed patterns from the FP-tree. To ensure that closed frequent patterns are being generated, we can verify the closed property using some hash based method.

TID	Items Bought	Frequent Items
100	*f,a,c,d,g,i,m,p*	*f,c,a,m,p*
200	*a,b,c,f,l,m,o*	*f,c,a,b,m*
300	*b,f,h,j,o*	*f,b*
400	*b,c,k,s,p*	*f,c,b,p*
500	*a,f,c,e,l,p,m,n*	*f,c,a,m,p*

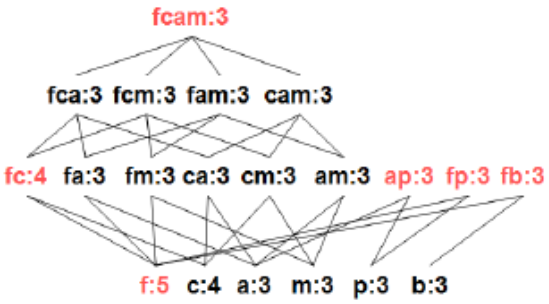

Figure 5: Closed Patterns

7. Compressed Frequent Pattern

Compressed Frequent Pattern Mining is a mining task where a representative Pattern is found for a subset of the complete frequent pattern set. Typically frequent patterns are clustered using a tightness measure δ and a representative pattern is found for each of the clusters. Finding the minimum set of representative patterns is NP-hard. Figure 6 shows an example clustering and a set of representative patterns.

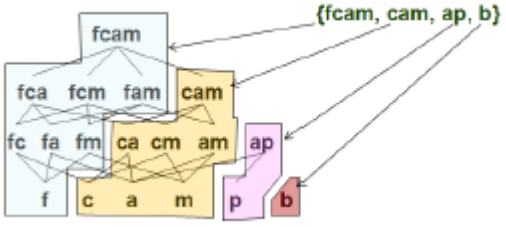

Figure 6. Closed patterns

Clustering the frequent patterns is done using the following distance measure.

$$D(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cap T(P_2)|}$$

Here P1 and P2 are two closed patterns and T(P) is the set of transactions that

support the pattern P. D is a distance metric. It satisfies Four properties of a metric- non-negativity, reflexivity, asymmetry and triangular inequality. The clustering is performed by finding a representative pattern Pr for each cluster where P$\subset$ Pr for every member P of its cluster and D(Pr,P) < δ Two algorithms for finding representative patterns are proposed. The RPglobal and RPlocal. The former is better in tightness but less efficient than the later. RP-global finds the complete set of covered patterns for each pattern. Two patterns are said to cover each other if their distance is less than δ. Then it iteratively picks the maximum coveriung set and thus tries to reach minimum number of clusters.

8. Performance Study

In this section we demonstrate the quality and computational performance using the benchmarks of the frequent itemset mining dataset repository. The algorithm is written in JAVA. The methods to be compared are summarized as follows. In the FP-Growth package, we generate all the closed frequent patterns w.r.t. M(minimum_supp). In the RPglobal method, we first use FP-Growth to get all the Closed frequent itemsets with min_sup $\hat{M} = M \times (1 - \delta)$, then use RPglobal to find a set of representative patterns covering all the patterns with min_sup M. In the RPlocal method, we directly compute all the representative patterns from database.

8.1 Number of Representative patterns

The first set of experiments comparre three algorithms w.r.t the number of output patterns. We select accidents, chess, and pumsb_star data sets. For each data, we vary the value of min_sup as the percentage of the number of total transactions and fix δ=0.1(we think it is a reasonably good compression quality). The results are shown in Fig.7 and Fig.8. We have the following observations: First, both RPglobal and RPlocal are able to find a subset of representative patterns, which is almost two orders of magnitude less than the whole collection of the closed patterns; Second, although RPlocal outputs more patterns than RPglobal, the performance of RPlocal is very close to RPglobal. Almost all the outputs of RPlocal are within two times of RPglobal.

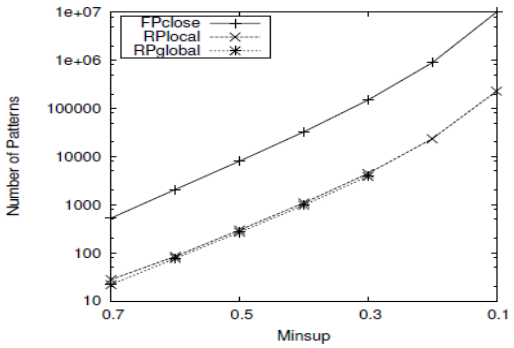

Figure.7 Number of Output Patterns w.r.t. min_sup, Accidents Data Set

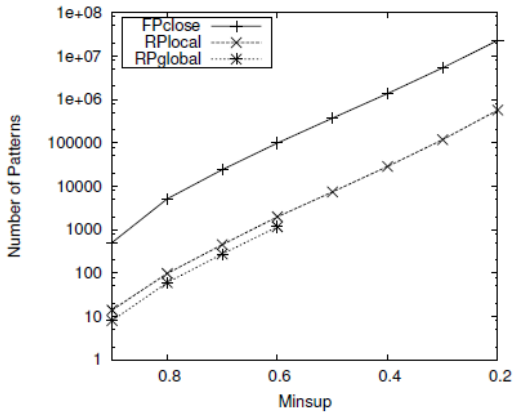

Figure.8 Number of Output patterns w.r.t. min_sup, Chess Data Set

9. Related Work

Lossless methods have been proposed to reduce the output size of frequent itemset patterns. [4] developed the concept of closed frequent patterns, and [5] proposed mining non-derivable frequent itemsets. These kinds of patterns are concise in the sense that all of the frequent patterns can be derived from these representations. However, they emphasize too much on the supports of patterns so that the compression power is limited.

Our work belongs to the family of Lossy

compression methods. Previous works in this direction include maximal patterns, error-tolerant patterns, δ-free itemsets and boundary cover sets. Typically, our work is close to error-tolerant patterns and δ-free itemsets. Our work is different in that we define a new distance measure and formulate the problem as set covering. Furthermore, we allow extended(i.e. longer) patterns to represent the compressed patterns, and it leads to stronger compression.

10. Conclusion

Mining frequent patterns is an important branch of data mining. Significant effort has been given to develop scalable and efficient algorithms. Besides other problems like sequential pattern mining, structured pattern mining, correlation pattern mining, etc are also being analyzed. Beside discovering frequent patterns, recent focus is on interpreting the frequent patterns more knowledgably. Newer criteria(e.g. timestamp, price) are being considered while mining the patterns. So frequent pattern mining is now established as a well studied and broad research area.

11. References

1. **Mining Frequent Patterns without candidate generation** – Jiawi Han, Hian Pei and Yiwen Yin
2. **Mining Top-K Frequent Closed Patterns without Minimum Support** –Jiawi Han, Jianyong Wang, Ying Lu and Petre Tzetkov
3. **Mining Compressed Frequent-Pattern Sets-** Dong Xin, Jiawi Han, Xipheng Yan and Hong Cheng
4. **Discovering frequent closed itemsets for association rules** -N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, , in Proc. ICDT'99, 398-416.
5. **Mining all non-derivable frequent itemsets :** T. Calders and B. Goethals, in Proc.PKDD'02, 74-85.